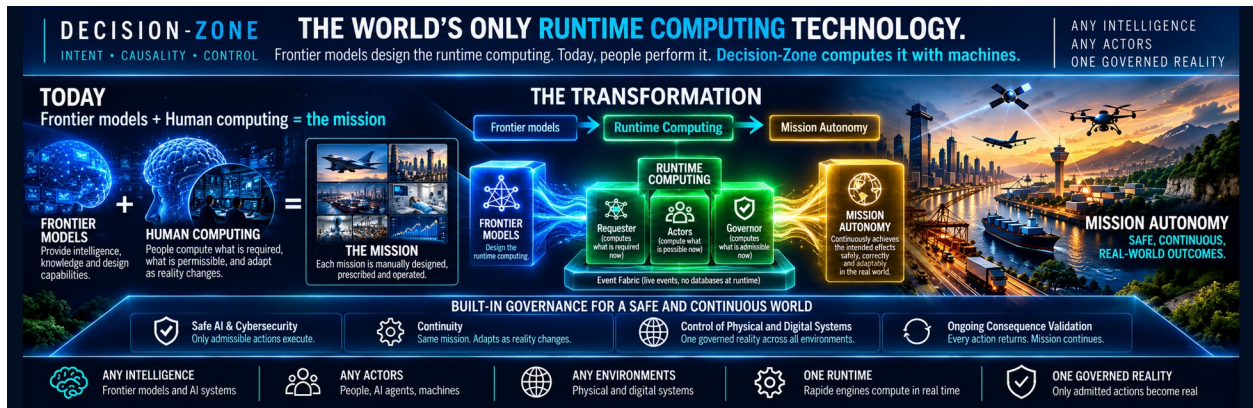




YOUR RUNTIME IS A PERSON

Why Software and AI Have Execution, Not Runtime Computation

An architectural paper on the computation that no system performs, and the one component that still does

THE CLAIM IN ONE PARAGRAPH

No conventional software system and no AI model has a runtime in the sense that matters. They have execution, which is the replay of decisions made at design time. Whatever has to be computed in the present is computed by a human being, and that human has never been named as part of the architecture.

The architecture that computes it instead is built, patented and running. This paper describes both.

**SOFTWARE EXECUTES WHAT WAS DECIDED EARLIER.
THE PRESENT IS STILL COMPUTED BY A PERSON.**

Triadic Causal Programming | Rapide runtime engines | U.S. Patent 7,908,260

ANY INTELLIGENCE. ANY ACTORS. ONE GOVERNED REALITY.

ABSTRACT

Written for the general reader. The sections that follow are for architects.

Every piece of software you use is a set of decisions somebody made in advance. A programmer thought about the situations the program would meet, decided what should happen in each, and wrote it down. When you run the program, it carries out those earlier decisions very quickly. That is what running software is.

This works beautifully as long as the world stays inside what was imagined. When it does not, the software keeps going anyway, confidently doing something that was right for a situation that is no longer happening. It has no way to notice.

So who notices? A person does. In every air traffic center, hospital, power grid, bank, factory and telecom network, there are people whose job is to watch what is actually happening, work out what is needed now, decide what is allowed, and keep things going when conditions change in a way nobody wrote down. We usually describe them as supervising the computers. They are not. They are performing a computation the computers cannot perform.

That job has never had a name, which is why nobody has ever tried to automate it. Its name is runtime computing. We automated the doing. We never automated the judging of what needs doing. Artificial intelligence has not changed this. A model is trained on an enormous record of the past and is extraordinarily good at producing answers from it, but a record of the world is not the world. Even an AI agent running in a continuous loop is reading a description of what just happened, not standing in what is happening.

This matters right now, because the technology industry is preparing to remove the people. Autonomous systems, agentic AI and self-managing infrastructure are all described as taking the human out of the loop. The assumption is that the human was there out of caution, and that better machines will earn the right to replace them.

That assumption is wrong, and the mistake is structural rather than cautious. The human was not supervising the system. The human was the part of the system that was present to reality. Remove them and nothing takes over the job. The machines carry on at full speed, with full confidence, executing decisions made for a world that has moved.

The missing piece has been built. Runtime computing is three computations kept independent of one another, running continuously against what is actually the case. One holds what is required. One holds what is possible. A third, separate from both, decides what is permitted. Nothing becomes real without passing that third one, and what actually happens is fed back so the requirement is worked out again. The programming model is patent pending. The runtime holds a granted United States patent and has been in development for twenty-five years.

Nothing in such a system is scripted. Boundaries are declared once, and conduct is computed continuously inside them. It is how a living thing works, and it is the only arrangement in which taking the human out leaves something behind.

1 | EXECUTION IS NOT RUNTIME COMPUTATION

The word runtime is doing two jobs, and the confusion between them has hidden a missing component for fifty years.

The first meaning is an execution environment. A JVM, a container, a process, an inference server. Something is loaded and instructions are carried out. Every software system has this and nobody disputes it.

The second meaning is runtime computation: something is being computed now, from what is actually the case now, that was not determined in advance. This is the meaning that matters, and almost nothing has it.

The difference is not pedantic, because the two have opposite relationships to time. An execution environment replays decisions made earlier. The code encodes what someone concluded at design time about what should happen under conditions they imagined. Running it faster, running it continuously, or running it in a loop does not change what is being replayed. A prescription executed at one millisecond is still a prescription.

Runtime computation has no prior conclusion to replay. It computes what is required, what is possible and what is permissible against conditions as they stand, and it does so again when they change. Nothing is retrieved, because there is nothing to retrieve. The situation has not occurred before in the form it now takes.

THE CLAIM, STATED PRECISELY

Not that software has no runtime. Software has execution and no runtime computation, and in every deployed system of consequence the runtime computation is still being performed, by a person.

2 | EVERY PRESCRIPTION HAS AN ENVELOPE

A prescription is not a flaw. It is a compression: someone thought carefully about a set of conditions and wrote down what should happen in each. Within those conditions it is faster, cheaper and more reliable than thinking again, which is the entire value of software.

The property that matters is the boundary. A prescription is adequate exactly as long as reality stays inside what was anticipated when it was written, and not one case further. Outside that envelope it does not degrade gracefully. It continues to execute, confidently, producing output that was correct for a situation that is no longer the one occurring.

This holds for every form prescription takes. A rule engine is prescription written as rules. A workflow is prescription written as sequence. A trained model is prescription written as weights, fitted to a distribution sampled in the past. A system prompt and a tool list are prescription written as text. In each case the question is the same: what happens at the edge of what was anticipated.

Two things can happen there. Either something computes in the present and works out what is now required, or the envelope is widened later by a person who noticed.

Software does the second. The widening is called a patch, a new rule, a retrain, a runbook entry or a change request, and it arrives after the event rather than during it.

WHERE THE INDUSTRY IS ACTUALLY SPENDING

The interesting question is not whether a system is intelligent. It is what happens in the gap between reality leaving the envelope and a person noticing.

Today that gap is filled by a person watching, and the whole discipline of operations exists to staff it.

3 | THE PERSON IS THE RUNTIME

Stand in any control room, network operations center, dispatch desk, flight operations room, trading floor or hospital command center and watch what the people are doing. They are not supervising the software. They are performing six computations the software does not perform.

1. Determining what is required now, which is not what was required an hour ago
2. Interpreting current reality from instruments that each report a fragment of it
3. Deciding what is permissible given authority, policy and circumstance
4. Coordinating which systems act, and in what order
5. Adapting when conditions change in a way nobody wrote down
6. Maintaining continuity across shifts, failures and partial information

Every one of those is a present-tense computation. None of them is retrieval. A person does them because a person is the only part of the arrangement that is actually present to the situation.

This is not a figure of speech. Being present means holding what is currently the case, continuously, and computing against it rather than against a description of it. A person in the room can see that what is in front of them is not what the system believes, and they act on the discrepancy. No component in the stack can do that, because every component has only the description.

The evidence is that nobody can remove the person

Every regulated industry mandates human oversight. Every autonomy roadmap keeps a human in the loop at the levels that matter. Every incident review ends with a person who noticed or a person who did not. These are usually explained as caution, immaturity or regulatory conservatism. They are none of those. They are the architecture reporting a load-bearing component, and the component is a human being.

WHY IT WAS NEVER AUTOMATED

The job had no name. You cannot automate a function nobody has written down. It has one now, and this paper writes it down: runtime computing.

4 | FOUR THINGS THAT LOOK LIKE RUNTIME COMPUTATION

Four classes of system are routinely offered as counterexamples. Each one responds to current conditions, and none of them computes a requirement.

System	What it genuinely computes now	What is still prescribed	Why it is not runtime computation
Closed-loop control (PID, MPC)	The correction needed to close the error against a setpoint	The setpoint itself, the control law, the bounds	It computes a correction, not a requirement. There is no admissibility and no completion, and the consequence does not return to a purpose. A control system holds a value. It cannot hold a purpose.
Event-driven and reactive systems	Which handler fires, and when	Every handler, every rule, every transition	The response to each event was written in advance. An event the author did not imagine has no handler, and the system does not notice that it should.
Machine learning inference	An output for an input never seen before	The objective, the distribution, the weights	Generalization within a distribution is interpolation over a record. It is the most sophisticated form of prescription ever built, and it is still fitted to the past.
Agentic AI	The next step in a loop, from a context window	The loop itself, the goal, the tool set, the escalation policy	Prescribed twice over. The observe-plan-act scaffold is written by a person, and each step is filled from a record. The context window is a report of what happened. The agent reads the report. It is never at the scene.

The pattern is one property with four disguises. In each case something adapts, and in each case what it adapts is a response, inside bounds set by a judgment made earlier, against a requirement nobody recomputed.

The agentic case deserves the extra sentence, because it is the one being sold as the solution. Speed is a property of the clock, not of the intelligence. Reading a record one second after the event rather than one day after does not change that a record is what is being read.

5 | TWO INTELLIGENCES

There are two kinds of intelligence at work here, and they differ in kind rather than in degree.

Design intelligence computes over a record. Its output is a structure. It reads everything that has been written, finds the pattern, and produces a plan, a model, a specification or a program. This is what a frontier model does, and it does it extraordinarily well. Understanding intent, discovering what systems exist and what they expose, mapping capability to effect, defining limits and compiling all of it into something coherent is genuinely the hard half of the work.

Runtime intelligence computes over the present. It has no corpus to consult and no prior case to match. Its output is not a structure but continuing adequacy to a situation that will not hold still.

A model cannot supply the second, and this is not a capability gap that scale will close. A model has no present. Its context window is a report. Something had to write down what happened before the model could read it, so by the time it reads, the moment is already over. The model is always reading about the world and never standing in it. Delivering the report faster does not put it at the scene.

THE CONSEQUENCE FOR THE MODEL RACE

Better models make design intelligence faster, cheaper and more available.

No quantity of design intelligence becomes runtime intelligence, any more than any quantity of maps becomes a journey.

This is why the two halves of the problem have had such different histories. Design intelligence has improved by orders of magnitude in a decade. Runtime intelligence has not improved at all, because it has never been built. It has been staffed.

6 | HOW THE PRESENT IS COMPUTED

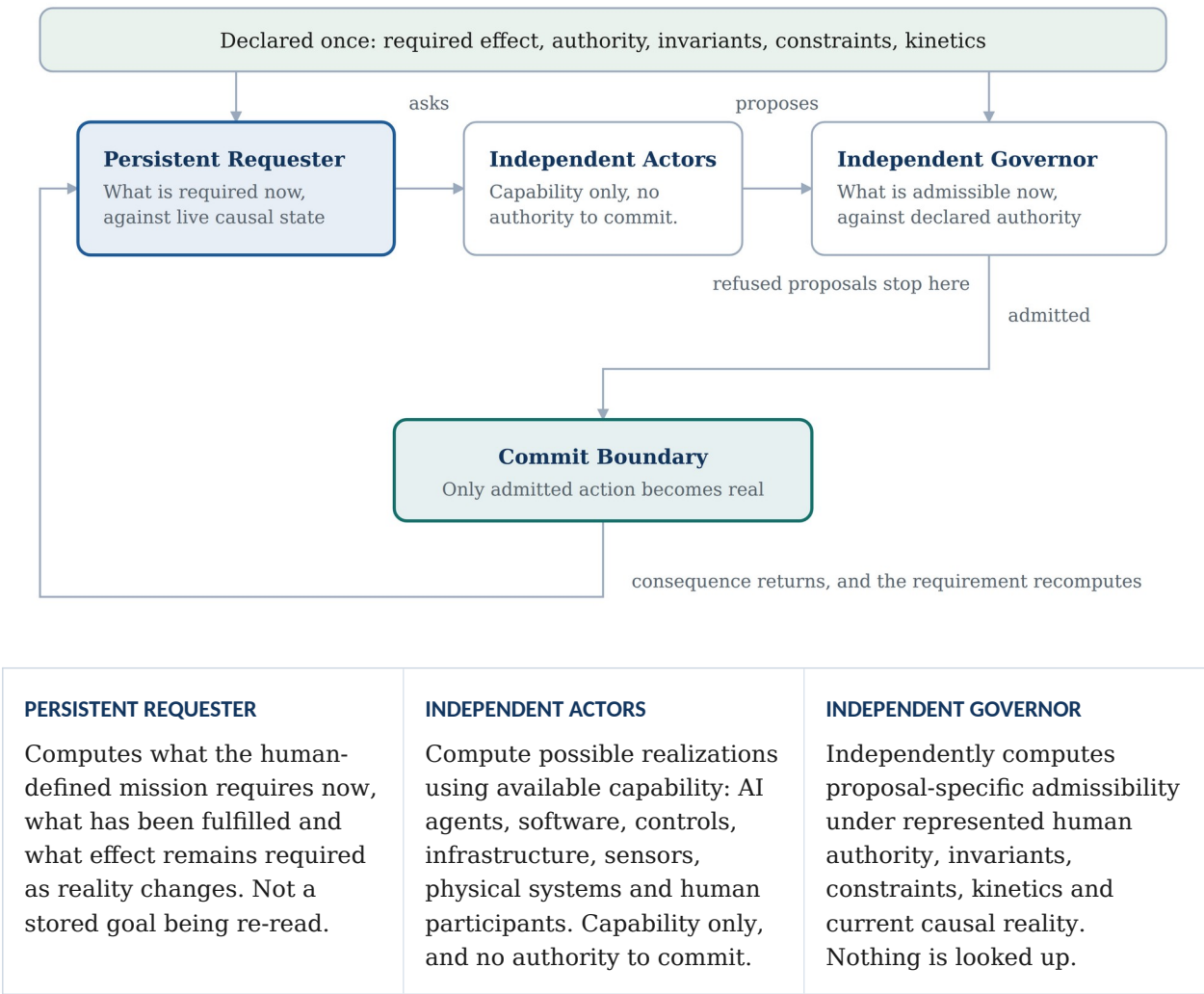
The structure that produces present is a closed loop of three computations that are independent of one another, running continuously against live causal state. Triadic Causal Programming declares that structure at design time. The Rapide runtime engines hold it while the mission runs.

FIGURE 1 | THE CLOSED TRIAD

The mission context is declared once. Requirement, possibility and admissibility are then computed independently and continuously against live causal state. Only an admitted proposal crosses the Commit Boundary, and the observed consequence returns to the same mission, which recomputes what remains required. Nothing about the interaction between the three is written down.

The intelligence is in the loop, not in any participant

Stop the loop and it is nowhere. Not in a file, not in weights.



REQUEST → PROPOSAL → DETERMINATION → COMMIT → CONSEQUENCE → RECOMPUTE

Only an admitted proposal crosses the Commit Boundary. Execution changes digital or physical reality. The observed consequence then returns to the same Persistent Requester, which recomputes fulfillment, remaining required effect, recovery or continuation.

What is declared is written once: the required effect, the authority, the invariants, the constraints and the kinetics. Boundaries, not behavior. What comes out is conduct

nobody authored, adapted to conditions nobody anticipated, produced continuously rather than loaded in at design time.

Two properties of the structure

The intelligence is a property of the loop rather than of any participant, which is why the Actors do not have to be intelligent for the system to be. And the intelligence exists only while the loop runs. Stop it and there is nothing to inspect. It is not in a file and it is not in weights, in the same way that a conversation is not in a transcript.

WHY THIS IS WHAT A LIVING SYSTEM DOES

Nothing in a living system is prescribed either. It has a persistent requirement, a body that can do certain things, constraints it cannot violate, and a continuous loop against whatever is actually happening.

The behavior is written nowhere. It emerges.

7 | THE SYSTEM THAT COMPUTES IT

The structure described in the previous section is not a proposal. It is built, it is patented, and it runs.

Design time: Triadic Causal Programming

TCP is the programming model. A mission is authored and compiled in it, declaring the Persistent Requester, the Independent Actors, the Independent Governor, the Commit Boundary and the consequence as independent computations, together with the invariants, constraints, kinetics, authority and event fabric the mission runs under. Nothing about the interaction is written: no sequence, no workflow, no handoff order, no exception path. A frontier model is very good at writing TCP, and during compilation it holds no authority and is not a source of intent. Patent pending.

Runtime: the Rapide engines

The runtime performs causal event construction, validation and execution, covered by granted U.S. Patent 7,908,260. It is built on the Stanford Rapide formal specification language and compiled into Java. Live causal state is held in memory over an Operational Event Fabric, with no databases at runtime and partial-order concurrency.

Admissibility is computed by backward checking, which is positive and deterministic: a precondition that is not confirmed resolves to inadmissible rather than to a probability. The runtime resolves each proposal to one of five outcomes: permit, block, suspend, escalate or revoke. This is formal determinism, not statistical confidence, and it is why the Governor can hold nothing between invocations.

TCP does not create, modify or alter a physical system or its intrinsic capabilities. It works through the interfaces machines already expose, including CAN, DDS, MQTT, REST, Modbus, robotics APIs and sensor feeds. Native control systems, safety mechanisms and embedded systems stay untouched and execute the admitted action themselves.

Simulation is not a separate tool. Rapide runs a compiled mission against simulated events first and against live events after, using the same engine, so a mission is exercised against reality before anything commits.

Demonstrators

Six mission demonstrators run the same architecture against different realities, with different Actors. The architecture does not change between them. Only the declared mission and the available Actors change.

Demonstrator	Mission	Scale and domain
CIP-40	City Infrastructure Mission	A fleet of 40 or more drones operating over city infrastructure
PIM-500	Portfolio Invest Mission	Hundreds of clients, each with its own persistent requirement
CPM-001	Continuous Patient Mission	Nine clinical scenarios
HM-001	Humanoid Mission	Governed humanoid operation
AEM-001	Agentic Enterprise Mission	Safe AI demonstrator, agents operating under an independent Governor
DMM-001	Defense Mission Autonomy	Counter-UAS and integrated defense, run against a live synthetic event stream

DMM-001 is the clearest view of the loop in operation. The Persistent Requester computes required effect as the event stream changes, Independent Actors propose realizations, the Independent Governor determines admissibility per proposal, and observed consequence returns to the same mission, which recomputes what remains required.

WHAT IS ALREADY SETTLED

U.S. Patent 7,908,260 is granted. The runtime is implemented and running. The programming model is filed. Twenty-five years of independent development stand behind both.
What remains is deployment into customer missions, not invention.

8 | THE CYBERSECURITY COROLLARY

One domain shows the general claim at work, and Decision-Zone published it separately a month before this paper. Cybersecurity is the clearest case because the attacker and the defender are both becoming computational while the mission between them is not.

AI makes the cybersecurity Actor extraordinarily capable. It can discover a vulnerability, infer an attack path, generate a detection, isolate a host or propose a network response. Capability answers one of the three questions an autonomous operation requires: what is possible. It leaves the other two to a human analyst, which is the same arrangement this paper describes everywhere else.

The consequence of computing the other two is specific and testable. The Governor does not need the ransomware Actor to identify itself as ransomware. It computes whether the proposed consequence, a mass encryption or a destructive modification or a privilege change, is admissible for this mission under represented authority and current causal reality. If it is not, it does not cross the Commit Boundary.

THE SECURITY QUESTION CHANGES

From: can we recognize every attack before it succeeds?

To: what may this Actor cause now, for this mission, under this authority, in this causal reality?

The same computation covers the cases that defeat recognition. A compromised AI Actor can generate a sophisticated proposal and cannot generate its own admissibility. An authenticated insider has legitimate access and still cannot establish what that access may cause now. A zero-day changes what an Actor is technically capable of doing and does not change the mission's requirement or governing authority by existing.

None of that is a cybersecurity feature. It is what happens in any domain once requirement and admissibility are computed in the present rather than supplied by whoever is watching.

9 | WHAT FOLLOWS FOR SAFETY AND AUTONOMY

Several things usually treated as separate problems turn out to be the same problem seen from different sides.

Safety stops waiting on alignment

If admissibility is computed by something independent of the thing proposing, the proposer does not have to be trustworthy. A model can be wrong, jailbroken or compromised, and its output is still only a proposal. What is bounded is consequence rather than cognition, which is a far weaker thing to have to guarantee and does not require anyone to solve alignment first.

This also inverts the usual security surface. In an actor-centric system authority is stored: an access control list, a policy store, a configuration, a retrieval index, a context window. Every published AI security failure is an attack on stored state read at decision time. If admissibility is computed now against live causal state, there is no stored authority to attack, poison or rephrase around.

Removing the human does not produce autonomy

In any deployment that puts agents in front of real systems, the person was supplying two things at once. They were the only part computing in the present, and they were

the only admissibility check that was not the proposer. Both are structural requirements of the architecture and neither has a name in it.

So the person cannot be removed by deciding to trust the agents more. Take them out and both functions vanish at the same moment, and nothing in the stack was built to assume either. The loop does not stop. It keeps emitting actions at the same rate and with the same confidence, and it cannot tell when its record stopped matching reality, because it has no present to compare the record against. A person at the console notices that what is in front of them is not what the system believes. An agent's confidence is identical in both cases.

THE WARNING, STATED PLAINLY

Removing the human from an agentic stack does not produce autonomy. It produces unsupervised prescription, and it removes the only governor that was ever in the building.

The autonomy ladder has the wrong rungs

Autonomy is usually described as a scale of trust: as systems get better, we let them do more. That framing makes the human a temporary measure awaiting sufficient confidence. The human is not a confidence problem. The human is a component.

HIGHER AUTONOMY DOES NOT ARRIVE WHEN THE MODELS GET GOOD ENOUGH TO TRUST. IT ARRIVES WHEN THE PRESENT IS COMPUTED BY SOMETHING OTHER THAN A PERSON STANDING THERE.

10 | HOW TO TEST ANY SYSTEM

Three questions settle it for any architecture, in any domain. They can be asked in a meeting and they cannot be answered with a roadmap.

Where is the requirement recomputed when conditions change?

If the answer is that someone notices and adjusts a setpoint, a rule, a threshold, a prompt or a plan, that person is the runtime. If the answer is that a stored goal is re-read each cycle, the goal is a prescription and nothing is being recomputed.

What can refuse an action, and is it the same thing that proposed it?

If the checker is another instance of the proposer, a guardrail trained the same way, or a model grading its own output, there is no independent admissibility. There is a second opinion from the same source. Independence is a structural property, not a matter of training a better checker.

When the episode ends, where does the mission live?

If the answer is that the context is summarized, the session closes, or the plan is marked complete, then nothing persists that could recompute what is still required. Continuity was being supplied by whoever was watching.

THE SHORTEST FORM OF ALL THREE

Show me what in your system is not written in advance and not derived from a record.

11 | SCOPE OF THE CLAIM

It is not claimed that prescription is wrong. Most software should be prescribed, because most situations are anticipated and compression is the point.

It is not claimed that models are inadequate. They supply design intelligence, they supply it well, and better ones make the design half faster and cheaper.

It is not claimed that the model is made safe. It is not. What is claimed is that an unsafe model stops being able to produce an unsafe action, which is a different and far more achievable guarantee.

What is claimed

- Runtime computation is a distinct thing from execution.
- It is currently performed by people in every deployed system of consequence.
- It can be computed by a closed triad of independent computations over live causal state.
- Until it is, every autonomy program is proposing to remove the only component that was doing the job.

Decision-Zone Inc. is a Canadian company. Once a mission is compiled it runs without any model in the path, which is what makes it deployable where a dependency on a model vendor is not acceptable.

12 | CONCLUSION

The industry is preparing to remove human beings from systems in which those human beings are the only component computing in the present and the only admissibility check that is not the proposer. It is doing so on the belief that the human was there out of caution.

The human is there because the computation exists and nothing else performs it. Name the computation and it becomes possible to build it. Build it and the human is freed to define more missions rather than being the loop inside each one. Leave it unnamed and every autonomy roadmap is a plan to run faster with nobody present.

**THE NEXT BREAKTHROUGH IN AUTONOMY MAY NOT BE
A MODEL THAT UNDERSTANDS MORE.
IT MAY BE AN ARCHITECTURE THAT IS PRESENT.**

**SOFTWARE EXECUTES WHAT WAS DECIDED EARLIER.
DECISION-ZONE COMPUTES WHAT IS REQUIRED NOW.**

ANY INTELLIGENCE. ANY ACTORS. ONE GOVERNED REALITY.

ARCHITECTURAL REFERENCES

U.S. Patent 7,908,260, runtime causal event construction, validation and execution. Method and System for Runtime Computing, patent pending, covering Triadic Causal Programming. Demonstrators: CIP-40, PIM-500, CPM-001, HM-001, AEM-001, DMM-001.

DECISION-ZONE

INTENT • CAUSALITY • CONTROL

TWO TYPES OF INTELLIGENCE

Different in Kind, Not Degree

ANY INTELLIGENCE
ANY ACTORS
ONE GOVERNED REALITY

Models supply one. The other has never been built, until now.

1 | DESIGN INTELLIGENCE

Computes over a record. Produces a structure.



OUTPUT: A STRUCTURE

Defines what the system is for and the boundaries within which it operates.



REALITY DOES NOT STAY STILL

The world changes, often in ways no one imagined earlier.

Supplies the design

Runs the mission now

- New events
- Unanticipated conditions
- Degraded capability
- Conflicting objectives
- Ambiguous information

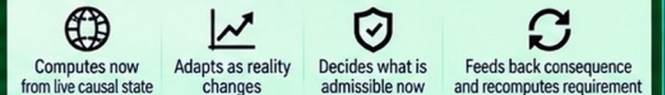
2 | RUNTIME INTELLIGENCE

Computes over the present. Produces continuing adequacy.



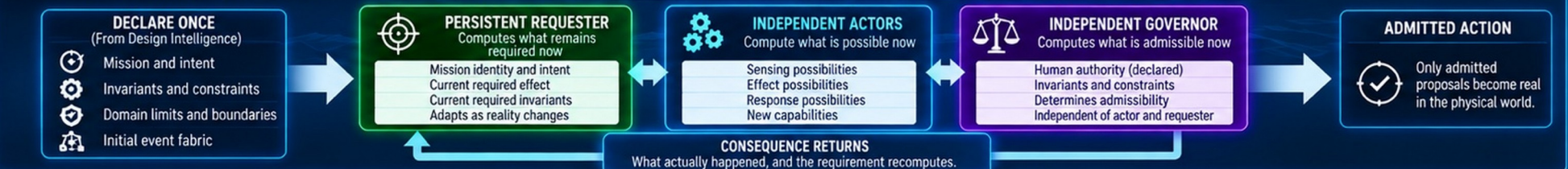
OUTPUT: CONTINUING ADEQUACY

Nothing becomes real without passing the independent governor.



3 | THE TRIADIC CAUSAL RUNTIME ARCHITECTURE

The first architecture that produces intelligence with no stored component.



WHY A MODEL CANNOT SUPPLY RUNTIME INTELLIGENCE

- ✗ A model has no present, only a record of the past.
- ✗ A context window is a description of the present, not the present.
- ✗ It cannot tell when its record stopped matching reality.
- ✗ Its confidence is identical whether it is right or wrong.

REMOVING THE HUMAN DOES NOT PRODUCE AUTONOMY

- ✗ It removes the only independent governor in the system.
- ✗ Agents continue to execute with confidence, even when reality has changed.
- ✗ The result is unsupervised prescription, not autonomy.
- ✗ Autonomy requires a governor that is independent of the actor.

WITH THE TRIAD, INTELLIGENCE EXISTS ONLY AS RUNNING

- ✓ What remains required, what is possible, what is admissible, all computed now against the live causal state.
- ✓ Nothing is scripted and nothing is retrieved.
- ✓ Stop the loop and the intelligence is nowhere.
- ✓ Not in a file, not in weights. It exists only as running, which is what we mean by self.

SOFTWARE AND AI HAVE EXECUTION. THE PRESENT IS STILL COMPUTED BY A PERSON.